

Optical Music Recognition: Feature Identification

David Bainbridge
Department of Computer Science
University of Canterbury

TR-COSC 02/95, Jan 1995

The contents of this work reflect the views of the authors who are responsible for the facts and accuracy of the data presented. Responsibility for the application of the material to specific cases, however, lies with any user of the report and no responsibility in such cases will be attributed to the author or to the University of Canterbury.

This technical report contains a research paper, development report or tutorial article which has been submitted for publication in a journal or for consideration by the commissioning organization. We ask you to respect the current and future owner of the copyright by keeping copying of this article to the essential minimum. Any requests for further copies should be sent to the author.

University of Canterbury
Department of Computer Science

**Optical Music Recognition:
Progress Report 2**

David Bainbridge
Supervisor: Dr. Tim Bell

January 6, 1995

Contents

1	Introduction	1
2	Separation of Staff and Musical Features	1
2.1	Deformation	2
2.2	Optical Character Recognition	5
3	Primitive Identification	6
3.1	Types of Pattern Recognition	8
3.1.1	Template matching	8
3.1.2	Hough Transform	11
3.2	Fragmentation Resolution Deferred	12
3.3	Results	14
3.3.1	Cmnxprim	14
3.3.2	Medxprim	16
3.4	General Points	18
4	Collating Data About Primitive Musical Shapes	19
5	The Drawing Package	20
6	Enhancements to the Prototype OMR System	20
6.1	Links to Existing Musical Packages	21
6.2	Musical Semantics of Text	21
7	Miscellaneous	21
7.1	A Hymn Database	21
7.2	The IMS Handbook of Musical Codes	24
8	Reflections and Future Thoughts	24

1 Introduction

Although it has been less than a year since the last progress report, work has reached a natural ‘break-point’ offering an opportunity to describe what has been accomplished as well as gathering thoughts on the future.

For a broader picture of how the implemented work fits into the general design, the reader is directed towards [Bai94a], which describes a complete Optical Music Recognition system. Since it is possible for a piece of music to include arbitrary graphics¹, it is impossible to design an OMR system that can process *all* music. The key idea, therefore, expressed in [Bai94a] is to provide a versatile foundation that can be built upon by individual users to generate particular instances of the system capable of recognising a particular class of music notation. Such a philosophy is reminiscent of Computer Aided Design (CAD). Similar to this area, the user should be encouraged to utilise sound software engineering principles. The proposed system could be thought of as a Computer Aided Music Recognition (CAMR).

The main body of this report describes the implemented work. The topics: staff separation; primitive identification; primitive data tabulation; drawing package development; prototype² musical feature classifier extensions; and miscellaneous items are discussed in turn. The report concludes by reflecting on the completed work as well as contemplating how the remaining problems may be solved.

Having the right development environment to study OMR is considered an important facet of work. Time throughout the project has been devoted to this. In addition to the topics listed in [Bai94b], time has recently been invested learning Perl (a Practical Extraction and Report Language), Gofer (a functional language), and Lime (a music file format for graphical reconstruction). Developing Internet searching skills using ‘whois’, ‘archie’, ‘gofer’, and ‘mosaic’ has also proved invaluable. Finally, in lieu of the forth coming design of a musical knowledge expression language, time has been spent investigating the suitability of Lisp, Prolog and various public domain knowledge based systems.

The excerpt of music shown in Figure 1 is a recurring example used, throughout this report, to illustrate various points.

2 Separation of Staff and Musical Features

The majority of this work was carried out last year and is described in [Bai94b]. Since then, two extra stages have been added: the first aims to correct any deformation in the image that was introduced through scanning; and the second addresses the complication of text causing ‘ghost’ staves to be detected.

¹The importance of this requirement is recognised in music editing computer programs by providing mechanisms for drawing arbitrary graphics, effectively degenerating into drawing packages.

²The OMR system originally started as an Honours project, with its various additions, is referred to as the prototype system in this report.

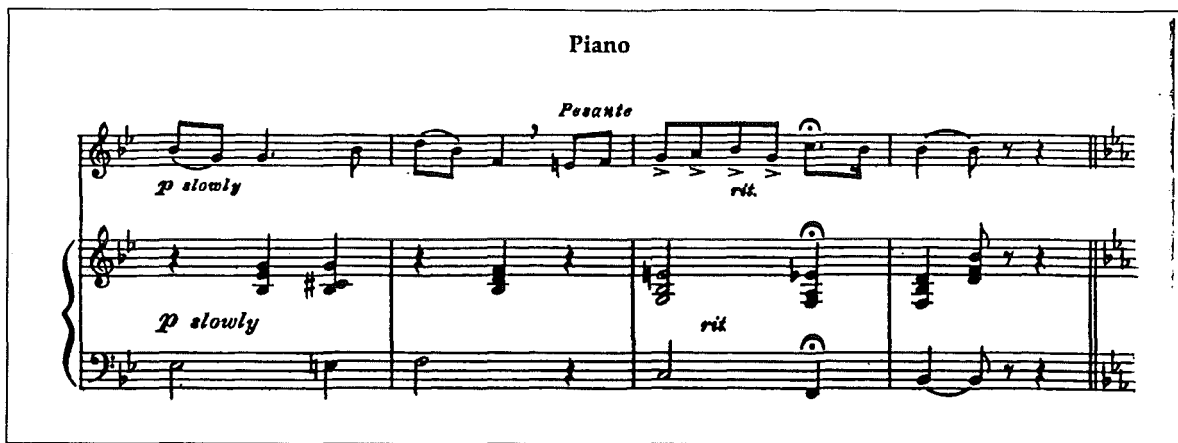


Figure 1: An example piece of music.

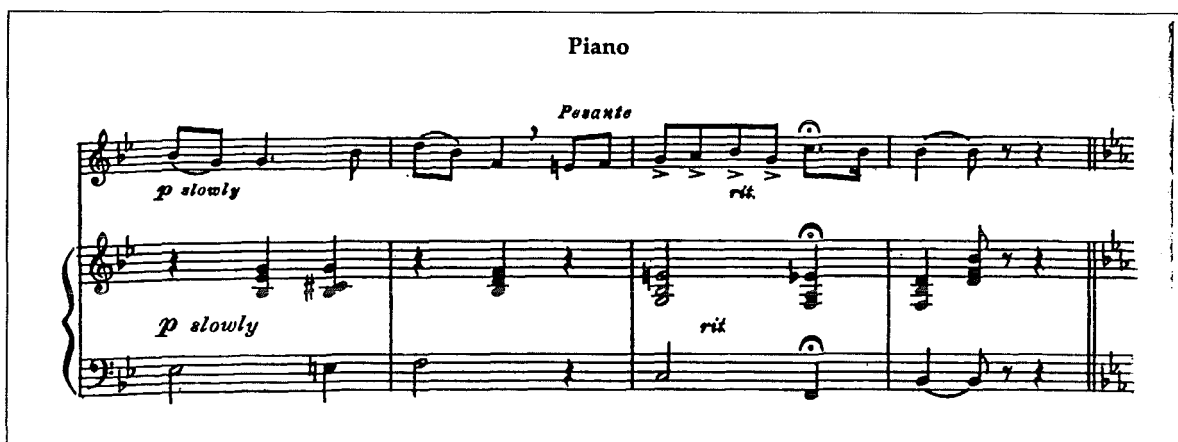


Figure 2: An piece of music intentionally deformed on the lefthand side.

2.1 Deformation

To study, in a controlled manner, the complications caused by deformation due to scanning, a program, **bmdeform**, was developed that *intentionally* deformed an image. A 'mint condition' scanned image and a freeform curve are presented to the program. The freeform curve defines a bijective mapping, continuous over the x-dimension of the scanned image. The program applies this curve to the image producing a distorted version. The cost of developing such a program was minimal since the C++ style of programming naturally promotes re-usable code.

The example image shown in Figure 1 was deformed with the intention of simulating a page scanned from a tightly bound book. The resultant image, deformed on the lefthand side, is shown in Figure 2.

Theoretically the staff lines in a piece of music are horizontal. Any deviation in the vertical direction, therefore, existing in the actually staff lines present in a scanned image, explicitly defines the deformation of that sliver of the image. In a typical piece of music, the extent of staff lines collectively cover a large proportion of the image. Accordingly, staff lines offer a convenient way of detecting the deformation

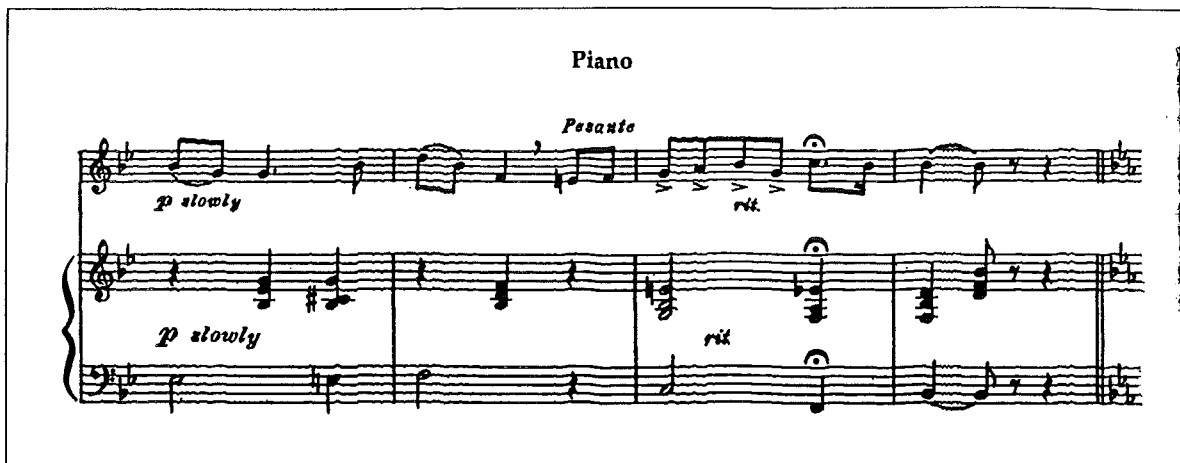


Figure 3: An image that has been corrected for deformation.

present in the scanned image.

One way to infer the distortion throughout the image would be to extrapolate the staff lines by 'morphing' between them. It is hoped that such a sophisticated algorithm will prove unnecessary. At present, a simpler idea has been encoded.

Staff location and staff removal were originally designed to accommodate, but not correct, deviations in the image due to deformation. Correction for deformation, therefore, is invoked after these two stages have been completed.

In the implemented algorithm, the located staff lines are collapsed in the y-direction into an accumulator. This 'generalised' staff line is then subjected to thresholding and thinning. Next any gaps in the line are filled and local unevenness smoothed out. Finally, the accumulated processed bitmap line is translated into a discrete mapping, which is applied, globally, to the scanned image. The consequence of applying the algorithm to the image shown in Figure 2 is illustrated in Figure 3.

The improvement in the image is perhaps best demonstrated by comparing the horizontal projections of the deformed and corrected images. Their respective projections are shown in Figure 4 and Figure 5.

The implemented algorithm is suitable for correcting global deformation, such as bowing staff lines, however it does not correct local deformations, such as a crumpled section of a page. If the latter scenario were to prove a common problem in a realistic environment, then a replacement algorithm using extrapolation techniques should prove sufficient.

Both of the discussed algorithms for correcting deformation are generalised forms of shearing. If an image were scanned skew and distorted, then the algorithm to correct deformation would not only reconstruct the original piece, but it would also be deskewed. With this work it may prove possible to circumvent the 'Deskew' stage in *xsep*. As a cautionary note, it is pointed out that such a correction for skew relies on shearing and not rotation; errors in the vertical aspect would remain.

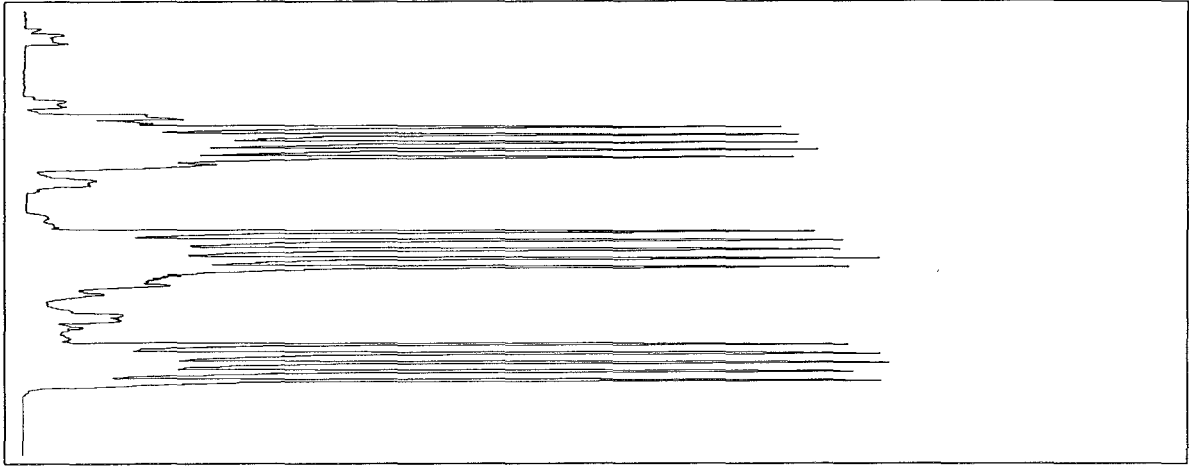


Figure 4: The horizontal projection of a deformed image.

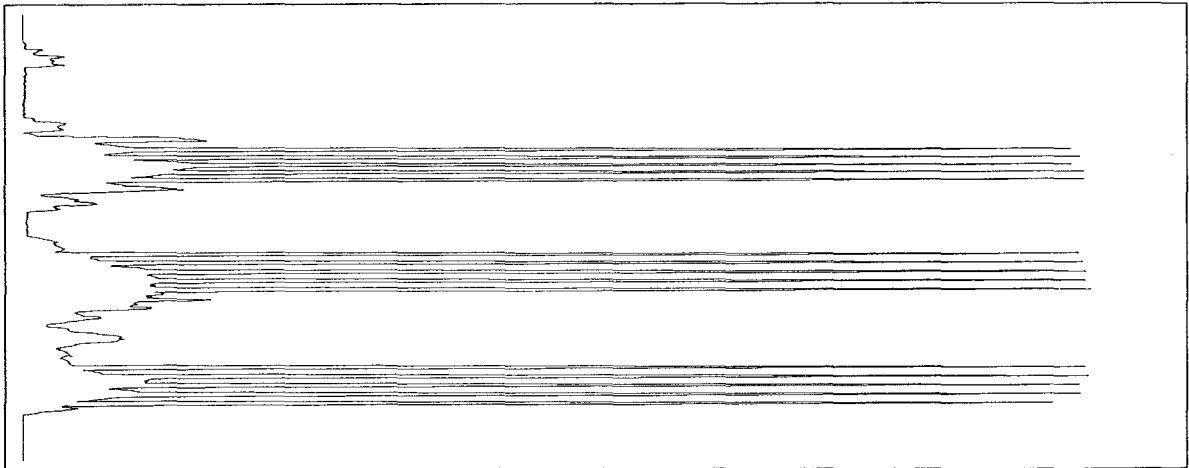


Figure 5: The horizontal projection of a image corrected for deformation.

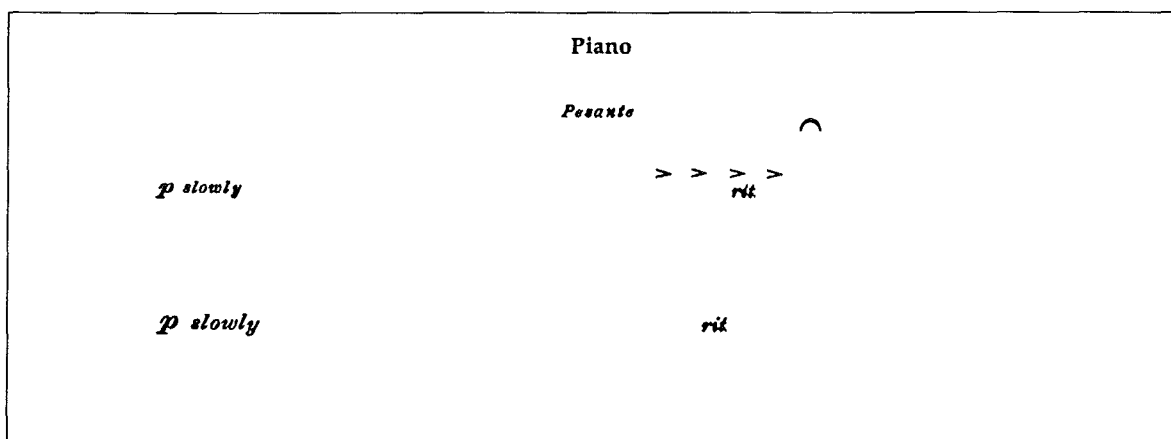


Figure 6: The located text in the image.

2.2 Optical Character Recognition

The algorithm developed to detect staves can misclassify a word in the image as a staff. Letters have three prominent levels where ink is present: the letter 'E' is the definitive example. In a horizontal projection, these areas accumulate, resulting in significant peaks, which are the cause of the incorrect classification. The problem is exacerbated by serifs which emphasize the thin horizontal peaks at the top and bottom of letters.

The problem was provisionally overcome by insisting the staves have more than three staff lines. However, this excludes the detection of percussion staves which consist of a single line, and the threshold of three is specific to the Roman alphabet. A better solution would be to identify and remove the text using an existing OCR package, prior to locating the staves.

OCR performed as the first step in the OMR process has repercussions, since the characters presented to the OCR package will no longer be corrected for skew. This is not as problematic as it may first appear. Depending on the sophistication of the OCR package, it may be able to accept characters that are skewed. If not, a simple alternative is available. Use the staff location algorithm to detect staves; this will include erroneous text. Since the skew of the text is identical to the skew of the staff lines, the angle calculated to correct the page can be used safely to level the image. The deskewed text can now be identified and the system continue as before.

The design on the OCR package is not the responsibility of the OMR designer. Ideally, there should be a 'black box' that is passed a bitmap and returns the closest matching character, with a certainty rating. The implemented system makes use of an OCR package written at Trinity College, Dublin. Although the package is not as sophisticated as its commercial counterparts, it forms a useful link, enabling an investigation of the requirements for text recognition within a musical score.

Text recognition was applied to the example music shown in Figure 1 as the first step. The result is shown in Figure 6.

When the recognition of text was originally added, it was introduced as a post-processing stage,

hence text characters were treated separately to the musically recognised features. Such a distinction is unnecessary and in fact undesirable. Letters in a page of music are equally valid musical features, conforming to 2-dimensional relationships and having musical significance. It is for this reason that it is acceptable for the accent marks in Figure 6 to be recognised by the OCR package, since all the musical primitives detected are treated homogeneously by later stages in the system. For the sake of completeness, it is mentioned that primitive naming may be hierarchical in the implemented system. It is achieved by assigning special significance to the colon character. For example, the letter 'E' may be named, 'text:E'. Completeness is ensured using the backslash, '\:' and '\\'.

Unfortunately a deficiency in the OCR package led to the incorrect classification of the 'slur' part of the pause marking in measure three from the example. Should the deficiency prove too restrictive, the OCR package will be enhanced to correct the problem.

3 Primitive Identification

In the prototype OMR system, primitive identification is combined with musical feature classification. A set of potential musical features is generated based on the current and surrounding bounding boxes, and the set of primitives that cover the set of musical features consequently derived. The system then takes the first named primitive, and searches the unknown object for its existence. Only musical feature candidates with the correct number of the detected primitive are kept. This process is repeated until there is only one musical feature left.

In the prototype system, the primitive identification module is controlled by the musical feature classification strategy. Since an explicit ordering of primitives exists in the system, the musical feature classification algorithm can improve performance by restricting the areas searched by the primitive identification module. For example, note stems are detected before note heads, hence the primitive identification module is instructed to only search areas to the left and right of the detected note stems.

An important question in OMR design is: what category of music do I wish to recognise? This covers not only the range of musical features that can appear, but also the quality of the scanned image. The top down methodology in the prototype system works well for the chosen range of musical features, however the approach is limited with respect to quality; the two main considerations are touching objects and fragmented objects.

Music often contains touching musical objects that should be separate. This can be caused by sloppy typesetting or an over-inked print run. In the prototype system, such an object would be recognised, at best, as only one of the musical objects, and at worst, neither.

Music can contain musical objects that are fragmented. The term fragmentation is commonly used in OMR literature to describe the problem of objects *becoming* fragmented due to the process of removing staff lines. Here the term refers to musical objects in a scanned page that are fragmented before any processing has started. An under-inked print run or a photocopied score can cause fragmentation, but an otherwise good quality piece may also include fragmented shapes due to a foreign element disrupting

the surface during printing or scanning. In the prototype system, it is rare for either of the objects to be recognised as the original musical feature.

The decision was made to replace the matching strategy with a bottom up approach. The new algorithm naturally deals with the complications of fragmented and touching objects. After the staff lines have been removed, the resultant image is searched globally for primitives. When an area of the page is checked for a primitive, no knowledge about how the isolated objects lie is assumed. Consequently for example, a break in a stem merely causes the match to be a few counts lower than the ideal match, and a sharp touching a note head would lead to some erroneous set pixels, also causing the match to be a few counts lower. Once all the primitives are identified, a new musical feature classification algorithm will combine them.

With this new design, primitive identification is an independent stage in the OMR process. It has been implemented as a separate program, **xprim**. The staff separation program, **xsep**, produces *sep* files which contain all the separated shapes. These files can be read by **xprim**, processed and saved as *prim* files which contain the identified primitives.

In the prototype system, it was the use of context to accelerate classification, that led to the restrictions concerning touching and fragments objects. Consequently the new design achieves its robustness by excluding the use of context at the expense of increased computational cost. Fortunately there are optimisations to the basic idea.

The task of identifying primitives is primarily interested in black pixels. A page of music, however, includes many white regions where no primitive can exist. Instead of checking the entire page, the isolated objects could be used to define the regions to check. Such an algorithm is proportional to the number of objects in the page, rather than the area of the page. This is no guarantee that the new algorithm will be faster, though intuitively one would expect so.

Another idea is to only test for a primitive if the origin of the region under consideration is set. In other words, the primitive must start with a black pixel. Unlike the first proposed modification, this change will not find exactly the same primitives as the basic algorithm, though the deviation should be negligible. Various combinations of the described algorithms were implemented. Table 1 presents the results.

Algorithm	User Process Time	System Process Time
Page	994199	3120
Set Page	154436	371
Objects	62187	381
Extended Objects	148724	470
Set Extended Objects	75017	350

Table 1: Comparison of bass clef curl identification algorithms.

For the isolated object based algorithm to faithfully reproduce the same results of the page based approach, it is necessary to expand the regions that are checked. Extending the region by the bounding box of the primitive that is sought, will cover the area where the primitive might exist. The extension

is needed exactly because of potential fragmentation of the primitive.

In Table 1 there is little relative change in the system times recorded. The role of displaying system time is to detect algorithms that are making unexpected demands on the kernel. On the other hand, values vary considerably for user process time. By far the worst time is the basic page algorithm. By restricting the check to locations that start with a set pixel, the algorithm is speed up by a factor of six (6.4376117). Checking regions defined by the isolated objects runs nearly 16 times faster (15.987248). However, as it has already been mentioned, if a primitive is likely to fragment, the extended version must be applied. This only runs seven times faster (7.1667412) since a considerable amount of time is spent checking the ‘empty’ space around the isolated shape. Worse still, when isolated objects are close together, duplicated areas are checked. The situation is improved by restricting the check to locations that start with a set pixel. This algorithm runs more like its non-extended counter-part, with a speed up of 13 (13.252983).

3.1 Types of Pattern Recognition

When designing pattern recognition routines, it is inevitable that empirical thresholds are used to decide if an unknown image matches a given shape. In the prototype system, such thresholds were static. As a wider range of musical material is considered, this becomes restrictive. In **xprim**, a mechanism for specifying thresholds dynamically is provided. In the *sep* file, an *emp* file can be specified. An *emp* file consists of statements of the form ‘*variable* = *value*’, where *variable* is the name of a threshold variable and *value* is its value. All threshold variables have a ‘reasonable’ default setting.

Another limit in the prototype system was its irrecoverable decision policy. Once a primitive was identified, it was assumed to be correct and removed from the image. This compromises robustness and reliability. The idea of uncertain data is not unique to OMR [BFK⁺92], nor is this the first time it has been proposed for OMR [FB92]. In **xprim**, uncertainty is introduced into the system by operating a two-tier threshold mechanism. A match above the upper threshold is a definite match (a certainty rating of 1.0) and consequently removed from the image. A match between the lower and upper threshold is assigned a proportional certainty rating between 0.0 and 1.0, and remains in the image.

Standard pattern recognition techniques are used to detect primitives. In addition to template matching, slicing techniques, and connectivity analysis utilised in the prototype system, the Hough transform [BT88] has proved useful.

3.1.1 Template matching

The term template matching describes a basic technique, variations are numerous [BMW94]. The prototype system relied solely on one implementation. In **xprim**, alternatives were investigated. The four algorithms were: standard, weighed input, averaged output, and weighted output.

The standard algorithm counted the pixels in the template that matched the unknown image. A percentage proportional to a perfect match was returned. The weighted input algorithm performed a correlation [GP87] between the template and the unknown image. Unlike the standard algorithm, the

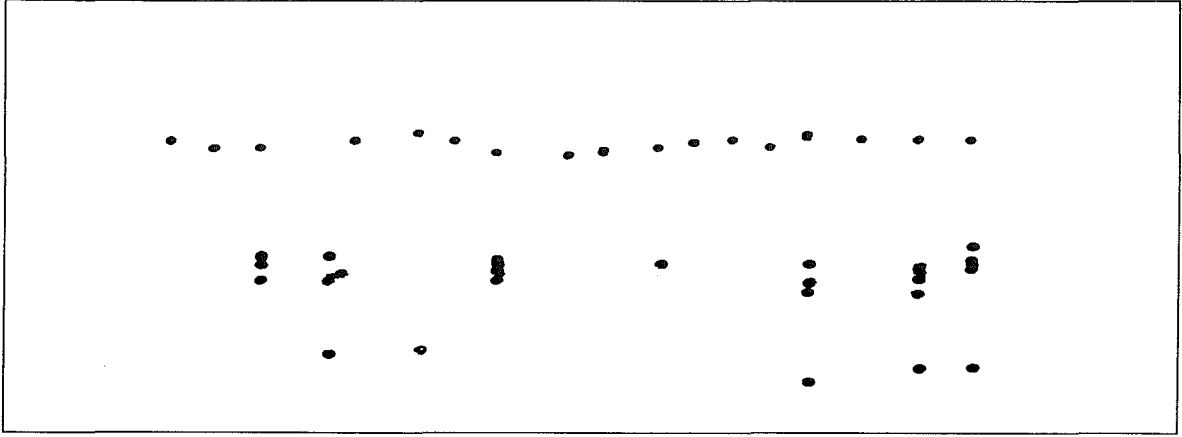


Figure 7: Full notes heads identified using standard algorithm.

template is no longer a bi-level mapping, but a grey-level mapping. When the template is designed, higher values are given to areas that are consider more important.

In the averaged output algorithm, if a template pixel does not match the corresponding pixel in the unknown bitmap, then the value is based on the average of correct matches that surround the pixel. If over half are correct matches, then the current pixel is recorded as a match. The motivation behind this algorithm is to reduce the effect of noise in the image. As a technical point, the area defined as 'surrounding pixels' is a rectangle, and the size can be varied within the program. Presently, a box of size three is used. The weighted output algorithm is similar to averaged output algorithm, except, when a pixel in the template does not produce a match, the local decision about a pixel being a match based on surrounding evidence, is not made. Rather, the number of of surrounding pixels that do match correctly is recorded for that pixel. When all pixels have been processed, the total 'similarity' count is compared against the perfect similarity count, and a percent match is computed accordingly.

To compare the algorithms, full note heads from Figure 1 were detected using each algorithm. The resultant images are shown in Figures 7 - 10. Full note heads that are drawn black have a certainty rating of 1.0; the grayed-out full note heads have a certainty rating somewhere between 0.0 and 1.0. With any of the full note head algorithms, hollow note heads that still have the staff line line passing through them can be detected as full note heads. Such bitmaps have negligible white pixels at the centre. A better algorithm would be to give increased importance to such pixels.

The threshold setting for one algorithm may not necessarily be the best for another. In Table 2 the computation times for the four algorithms along with the threshold setting are shown. With suitable settings for the thresholds, there is little difference between the identified full note heads. The computational cost, however, of the averaged output and weighted output algorithms is around three times more expensive than the standard and weighted input solutions. The difference is due to the former algorithms considering the information of surrounding pixels.

The template for the weighted input algorithm was designed to increase the importance of the central

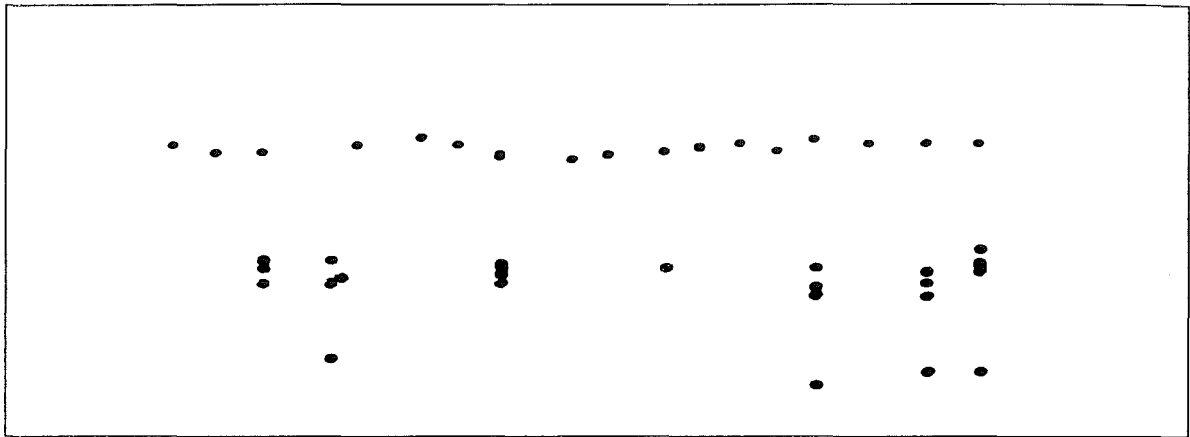


Figure 8: Full notes heads identified using weighted input algorithm.

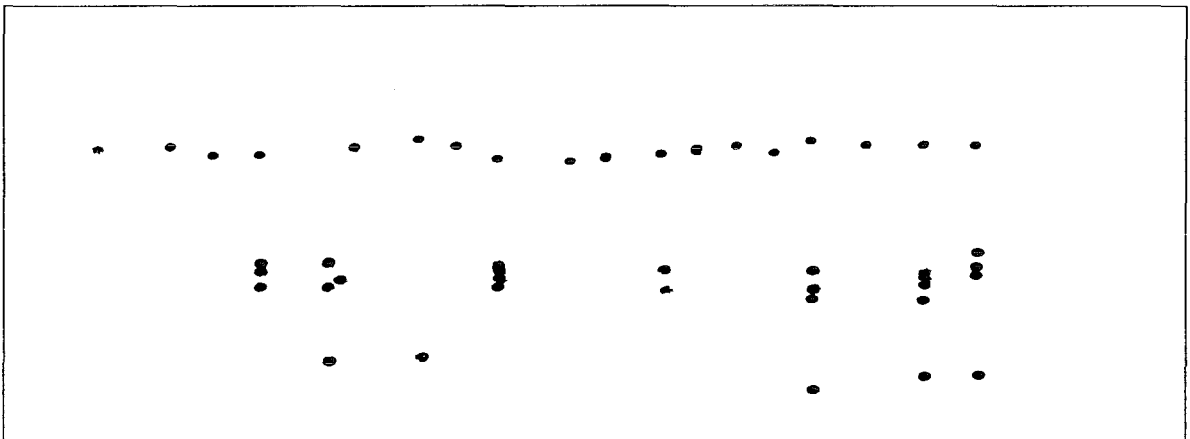


Figure 9: Full notes heads identified using averaged output algorithm.

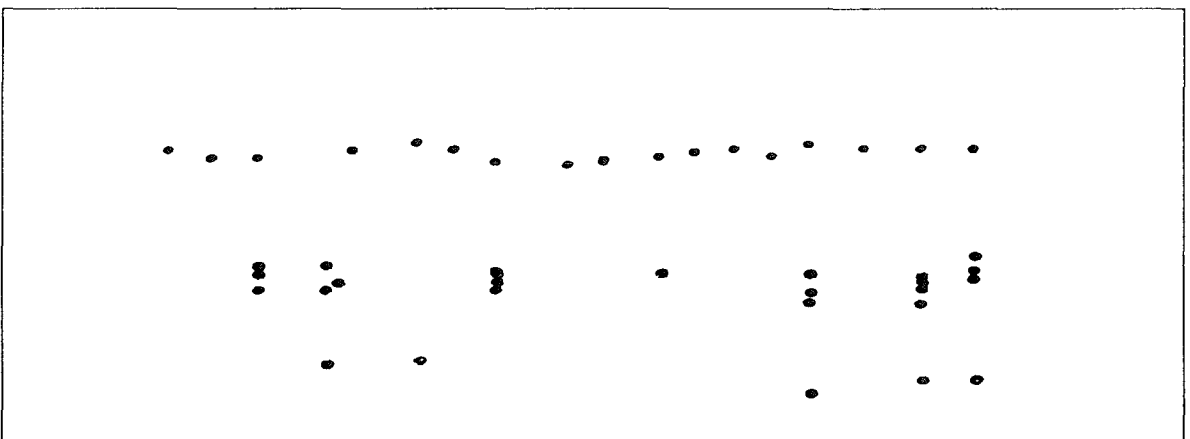


Figure 10: Full notes heads identified using weighted output algorithm.

Algorithm	User Process Time	System Process Time	Lower Threshold	Upper Threshold
standard	40695	180	80	85
weighted input	38133	181	92	95
averaged output	110231	220	85	89
weighted output	111431	350	88	90

Table 2: Alternative algorithms for identifying full note heads.

area of the shape. The design was motivated by the clusters of note heads that form a chord, intending to promote the clear separation of note heads. Unfortunately this template is prone to matching areas of the image that have a heavy central cluster of set pixels, but do not define an elliptical full note head. If the weighted input algorithm is to be used extensively, the design needs more consideration, say additionally weighting the perimeter to the shape. Clearly there is an art to designing weighted templates.

3.1.2 Hough Transform

The Hough transform is a method for evaluating the likelihood of a parametric equation existing at a specific place in the image. It is a technique commonly used in Computer Vision. The following description is based on explanation given in [BT88].

As the name suggests, the Hough transform works in a parameter space; the parameters in question are those used to describe the feature we are looking for. Naturally we are interested in the two-dimensional case — for this discussion the parameters a and b will be used.

An accumulator, A , is defined over the parameter space and all entries are initialised to zero. For each coordinate (x, y) in the image, all accumulator entries $A[a, b]$ that are a solution to Equation 1 are incremented.

$$(a, b) = (x + f(t), y + g(t)) \quad (1)$$

Where x and y are co-ordinates in the bitmap.

And $f(t)$ and $g(t)$ are the parametric version of the sought feature.

In most situations the range and increment of a and b are the same as x and y . A informal explanation of the transform would be: for each co-ordinate in the bitmap, all the positions of the parametric equation that could pass through this point are recorded in the accumulator.

The Hough transform is well suited for primitive identification, since it is unaffected by fragmentation and touching. Freeform curves [FvDFH90], which are (conveniently) parametric, have been found to be particularly useful. They have been successfully used to identify bass clef curls, hollow note heads and the ‘loop’ part of a flat. As a cautionary note, however, it is emphasised that the Hough transform is a relatively expensive operation and prudence should be exercised over its use.



Figure 11: Original bass clef.

3.2 Fragmentation Resolution Deferred

Removing staff lines fragments objects. Even if obvious precautions are taken, such as checking above and below the staff line for evidence of an object, the fragmentation of lines that blend tangentially into a staff line, still occur. Work by Martin and Bellissant, [MB91], described a chord based strategy aimed at keeping blending lines joined. An implementation by the author, based on the paper, found that although the algorithm further reduced cases of fragmentation, some tangentially blending objects became broken. Also, some objects that should not touch became joined. Both these situations were also reported by Martin and Bellissant. It may prove that adaptation to the algorithm will eliminate the remaining problems, however the author found the algorithm vastly expensive in processor time.

Since the problem of fragmentation is caused by staff line removal, it is tempting to believe the solution must also lie within the staff line removal algorithm. It is argued here that this is not the point to attempt complete fragmentation correction. It was pointed out earlier in the report that objects may be fragmented due to reasons other than staff line removal. Fragmentation issues must be faced by other stages in the OMR process.

In general, existing algorithms for correcting fragmentation tend to be heuristic rules which attempt to capture the types of situations that result from fragmentation. They are basically primitive shape driven, but they attempt to correct the image ahead of time i.e. before the primitives have been identified. Why not leave the correction process until primitives are being identified? For each type of primitive, there is a notion of the shape encapsulated in algorithm which detects that particular primitive. This information should be exploited to correct fragmentation. The design of `xprim` advocates this strategy. The Hough transform is particularly suitable.

Successful application of the idea is illustrated in Figures 11 - 13, which shows the correct identification of the bass clef from the original example.

As a final point, it is emphasised that, although substantial fragmentation correction occurs later on in the system, reducing fragmentation in the staff line removal algorithm should not be neglected.

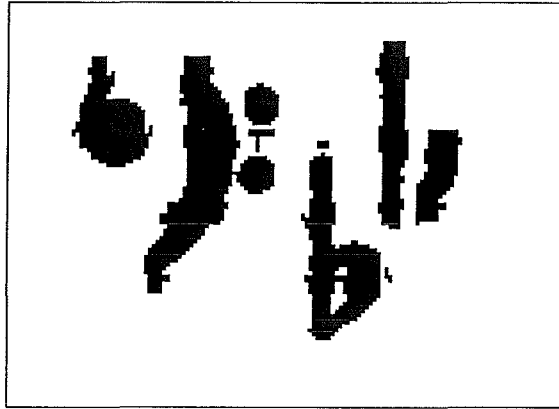


Figure 12: Isolated bass clef.

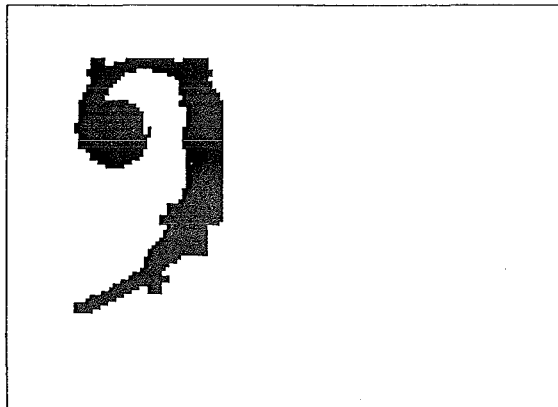


Figure 13: Identified bass clef primitive.

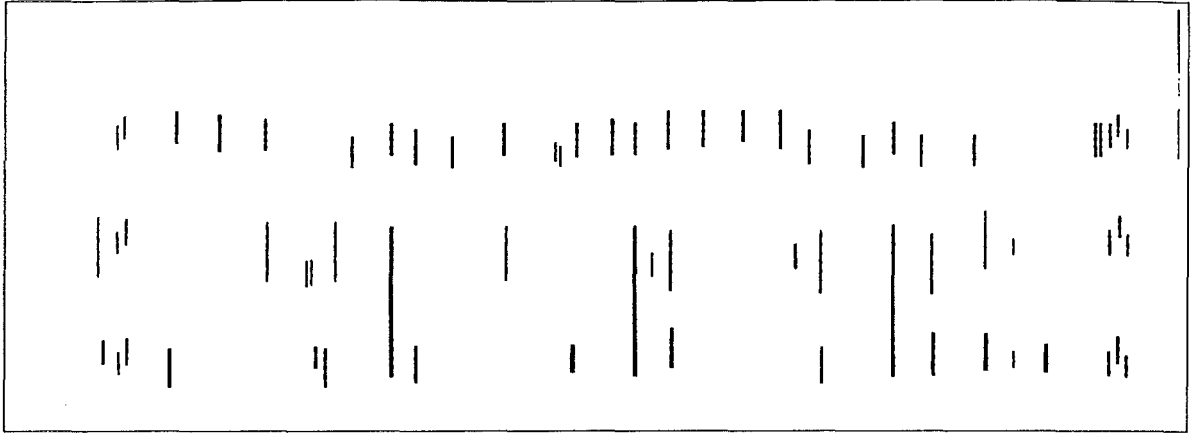


Figure 14: Identified vertical lines.

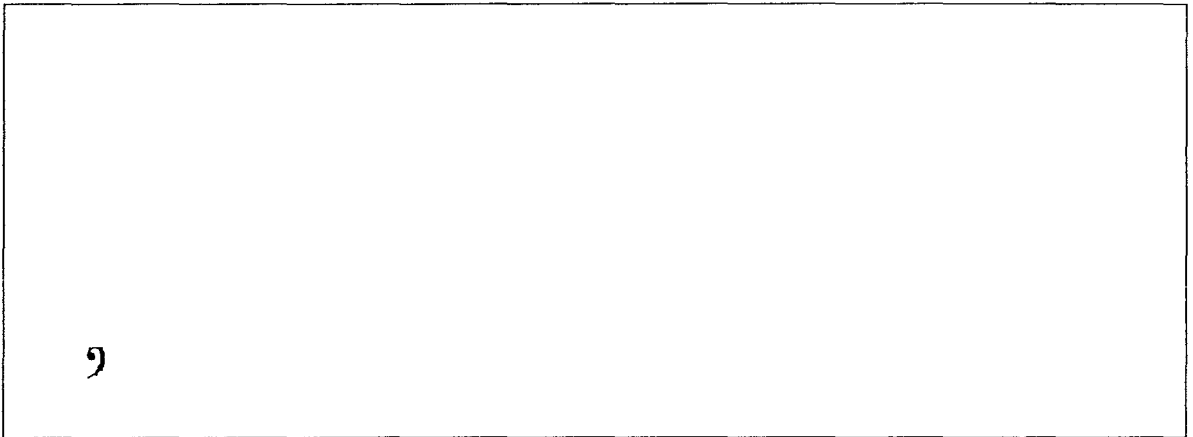


Figure 15: Identified bass clef curls.

3.3 Results

Primitive identification for CMN and Mediaeval music have been developed. The respective programs are called `cmnxprim` and `medxprim`. In the future it is planned to replace these two static programs with a single dynamic version, `xdprim`, which accepts a file describing the primitives that to be detected.

3.3.1 Cmnxprim

Figures 14 - 18 show the primitives found in the example piece of music, Figure 1. As the figures show, the system allows music with different sized staves. Two arrays, $staff_{min}$ and $staff_{max}$, ranging over the y-dimension of the image are maintained. Both arrays are initialised by setting entries that correspond to staff areas in the image, to the staff height found there, and zero elsewhere. Next the gaps (entries set to zero) are filled in with either the minimum or maximum enclosing region, depending on which array is being processed.

When processing an area of the image that is a staff area, there is only one size of musical features.

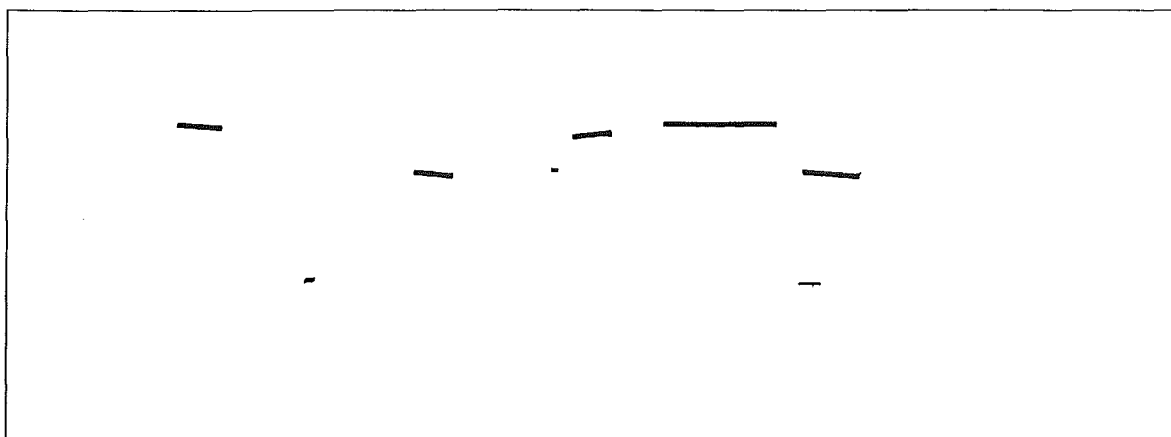


Figure 16: Identified beams.

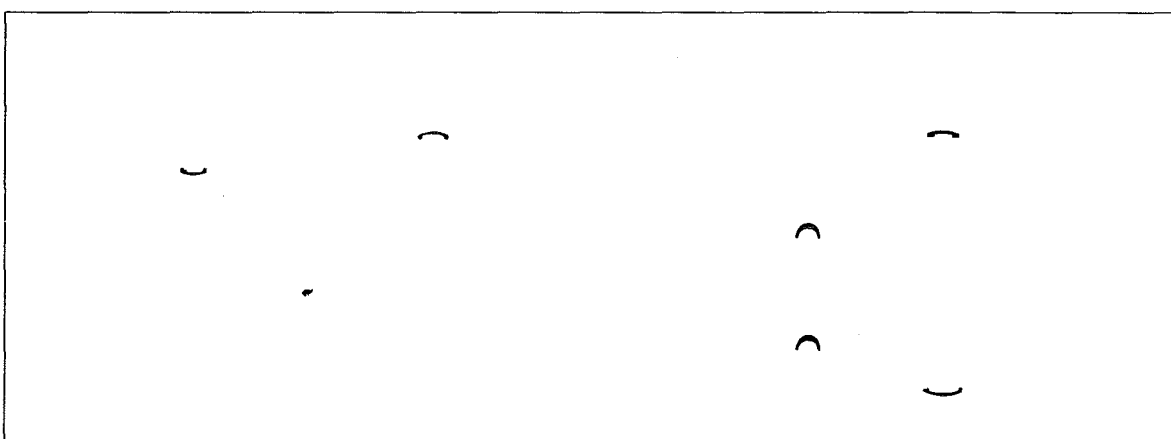


Figure 17: Identified slurs.

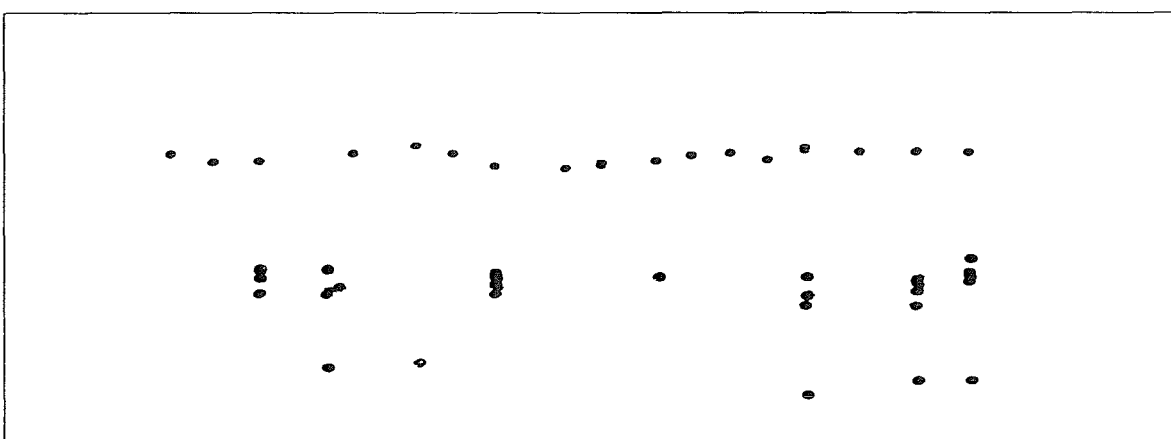


Figure 18: Identified full note heads.



Figure 19: Excerpt of Mediaeval music.

This information is readily available from either array. When processing an area between two staff area, there can be at most two sizes of musical features. This information is readily available from the two arrays. In principle, this algorithm permits an arbitrary number of distinct sized staves, however in practice music only has at most two sizes of staff, for example a piano accompaniment might have the solo part cued above it.

In Figure 14, the detection of vertical lines has failed to locate some of the shorter vertical lines as well as some of the longer vertical lines. Additionally many medium length length lines are classified as 'uncertain', when visually there is little doubt that they are vertical lines. Tuning the empirical thresholds will improve the result. It is also believe that there are improvements that can be made to the algorithm. For example, the longer the line is, the greater the chance of its bounding box widens. A better measure of an objects 'thinness' would be the average of its horizontal projection, rather than the y-component of the bounding box.

The comment about tuning empirical thresholds holds true for most of the primitive types. Currently these thresholds have been set at 'reasonable' values. Extensive testing is needed to iterate these values to hopefully stable settings. Also, the set of primitives searched for is not yet complete. Some incorrect classifications in Figures 14 - 18 would be removed by expanding the set.

3.3.2 Medxprim

An excerpt of a mediaeval piece of music is shown in Figure 19. The detected primitives are presented in Figures 20 - 23.

The printing process was less sophisticated in Mediaeval times, consequently tolerances on matching primitives in such images will need to be relaxed. For example the short vertical line that intersects the top line in the staff is not correctly identified. This is because it is wider than the permitted tolerance. Once again extensive testing has not yet been carried out.

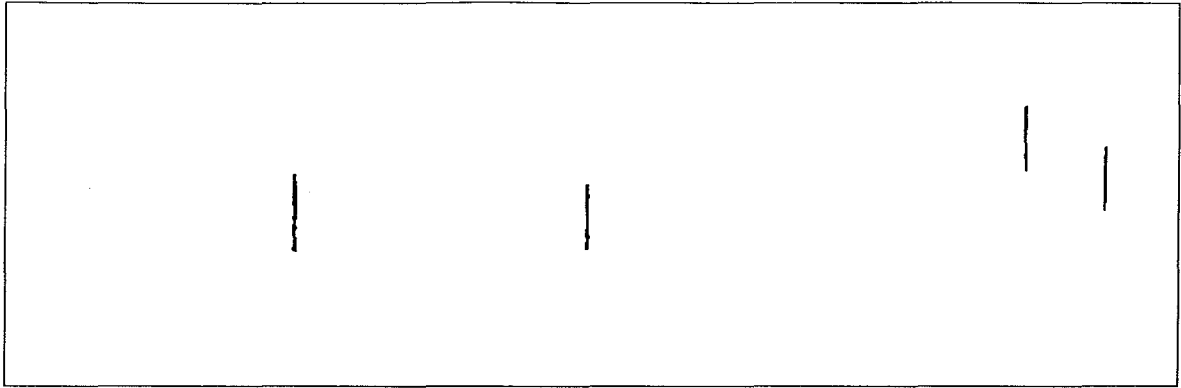


Figure 20: Vertical lines in excerpt of mediaeval music.

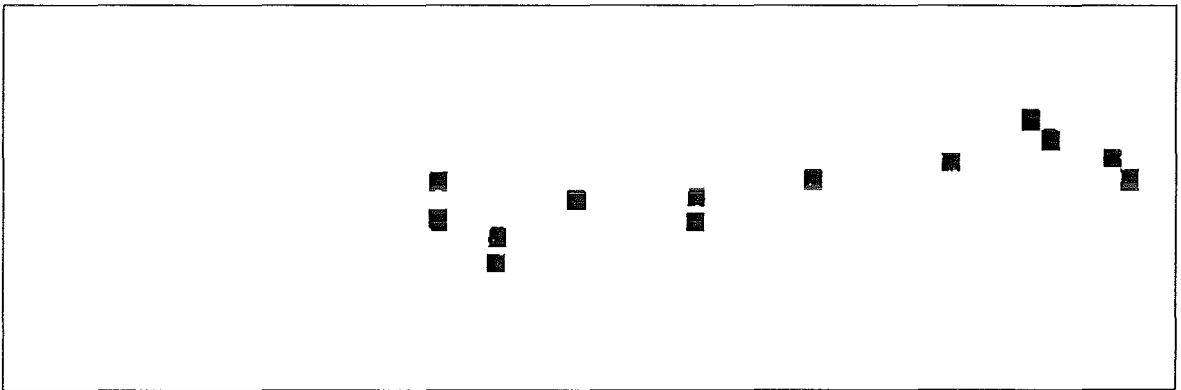


Figure 21: Rectangles in excerpt of mediaeval music.

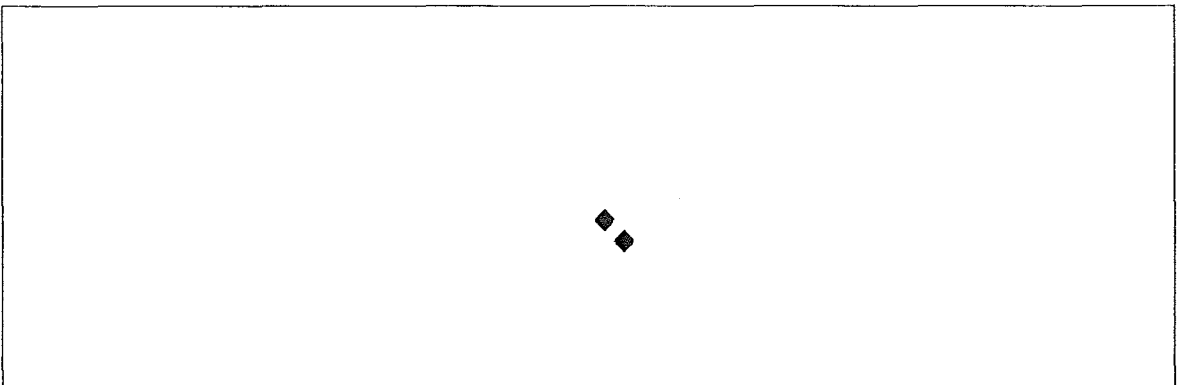


Figure 22: Rhombus in excerpt of mediaeval music.

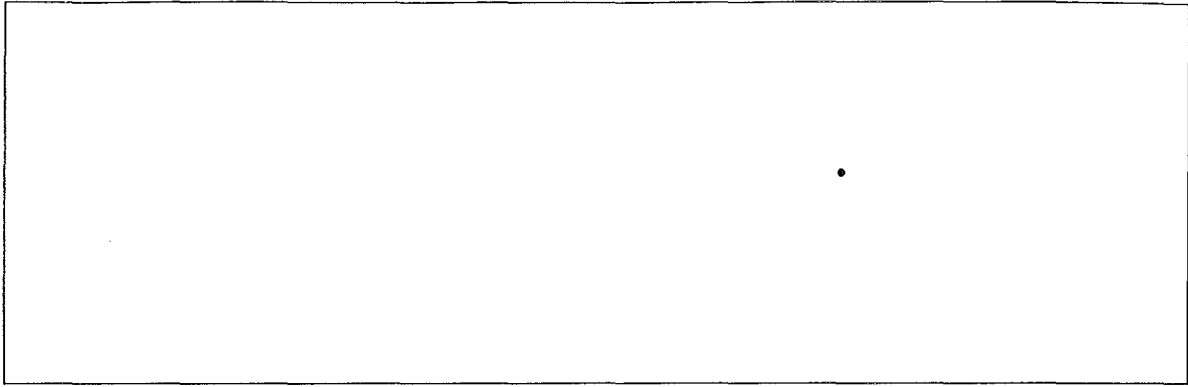


Figure 23: Dots in excerpt of mediaeval music.

3.4 General Points

Break-points were introduced into the prototype system to aid the investigation of classification decisions made about particular objects in the image. Often it is difficult to see why an object has been classified a particular way. Even if it were practical to include diagnostic output statements in the program to print out all the relevant information, this would occur for all objects, quickly obscuring the information about the desired object.

A better solution is to use break-points in conjunction with a debugger. When an object in the image contains the co-ordinate specified as the break-point, a 'kill' signal is generated. This is caught by the debugger and execution is halted. The code can then be stepped through and values printed out. In *xprim*, the idea of break-points is carried one step further. When a break-point is set, the primitives the break-point applies to are selected. The program will only halt when the primitive it is searching for one of the selected primitives and the object currently under consideration includes the break-point.

Independent primitive identification is a time-saving facility. Even though there is a specific order to search for primitives, and some primitives explicitly make use of this fact, a primitive does not necessarily require all the previous primitives to be detected. When developing an algorithm that detects a particular primitive, time may be wasted identifying primitives that do not effect the new primitives. Switching the option in the graphical user interface from *dependent* to *independent* has the effect of cloning a list of object without primitives *Y* from the list for primitives without *X*, for all stages between the last detected primitive and the one that is desired. In effect what happens is the program quickly generates the result of searching for primitive *Y*, when none are found.

From the few experiments carried out so far, it seems likely that the original expectation that accidentals could be decomposed into vertical lines, small beams, and loops, in a publisher independent way is too optimistic. The problem is that the beam part of sharps and naturals is prone to noise caused by removing staff lines. Consequently they are hard to recognise reliably. It is likely that the system will revert to using templates to recognise these musical features. Though unfortunate, the change is still within the bounds of the dynamic design requirement.

```
@filenames = ("scanned/burgh/excerpt/*.prim");
$prims = "vertical_lines";
$function = dimensions;
$gofer = "let x_coord (x,y) = x in sum (map x_coord list) / length list";
```

Figure 24: A collate file program.

```
Searching with the following settings ...

filenames = scanned/burgh/excerpt/*.prim
prims = vertical_lines
function = dimensions
gofer = let x_coord (x,y) = x in sum (map x_coord list) / length list

Expression constructed for gofer is ....

let list = [ (5,269), (6,53), (5,60), (5,58), (5,59), (5,58),
(5,124), (5,121), (6,123), (6,271), (5,68), (5,88), (6,112), (6,87),
(5,67), (6,271), (4,122), (5,122), (5,123), (5,122), (7,272), (5,73),
(5,125), (5,83), (5,69), (4,132), (5,84), (6,273) ]
in let x_coord (x,y) = x in sum (map x_coord list) / length list

Gofer returned ...

? 5 :: Int
```

Figure 25: Output from the collate program.

4 Collating Data About Primitive Musical Shapes

It has already been mentioned that a trial and error strategy is used to set the thresholds for primitive identification. Such a practice is common with OMR researchers. Though useful, it is not an entirely satisfactory arrangement. A desire to be more scientific in setting these values, motivated the design of **collate** — a program that tabulates the data stored in the *prim* files.

To run **collate**, the user specifies: a regular expression defining the files to be searched; a regular expression defining the valid primitives; the output function to call for each valid object; and a Gofer function to apply to the generated list of data.

A Perl script decides which files are to be read, and repeatedly calls a C++ program with suitable arguments to generate the data. The resultant list is then combined with the specified Gofer function and interpreted. Figure 24 is a simple example ‘program’, which computes the average width of vertical lines. When run, it generates the information shown in Figure 25. The choice of the programming language to process the data list is insignificant — any list processing language would be appropriate.

5 The Drawing Package

A vital attribute of the planned OMR system is dynamic primitives. Not only should thresholds be set dynamically, but the actual description of the primitives to be searched for, should be dynamic. A customised drawing package is an integral component to this design. An example of the current drawing package is shown in Figure 26.

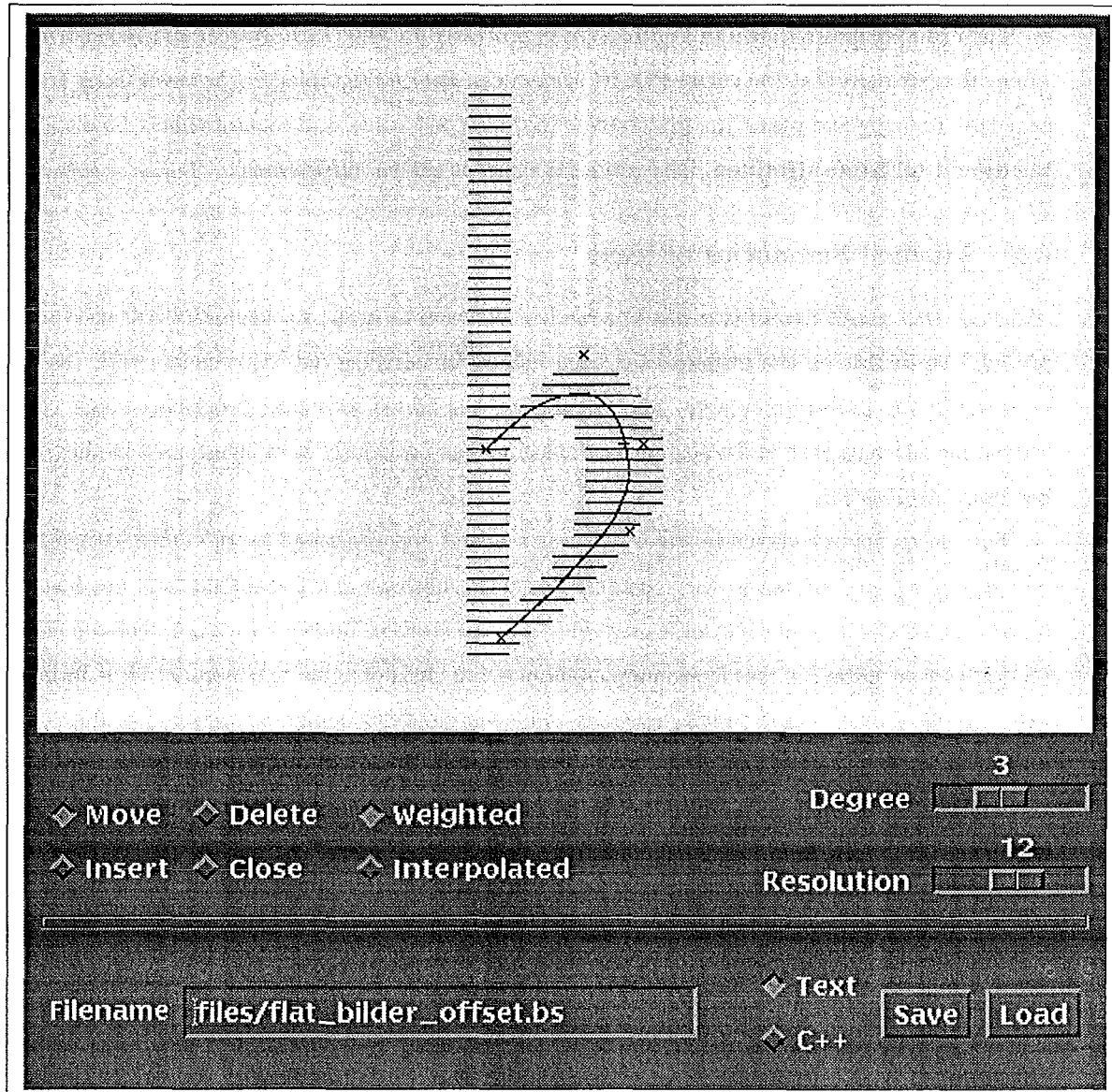


Figure 26: Example of using the drawing package.

6 Enhancements to the Prototype OMR System

The prototype program will eventually be replaced, however there is still much that can be learnt from extending it. It is also anticipated that code can be recycled in the new system.

6.1 Links to Existing Musical Packages

A module for generating MIDI [Rum90] and LIME [HB92] files from the musically recognised page was written. Additional utilities were developed that filtered and concatenated LIME file. As a consequence, valuable knowledge about the two musical file formats has been learnt, which will prove beneficial when developing corresponding routines in the dynamic system. It has also highlighted common programming requirements for musical file output functions.

The example music shown in Figure 27 was processed by the OMR system and reconstructed using Lime. It is stressed that the current OMR system can *not* detect tablature, however using Lime it is an easy task to ‘copy and paste’ the first voice into the second voice, and request that it be displayed using the default tablature algorithm. The resultant music is shown in Figure 28.

6.2 Musical Semantics of Text

Although the recognition of text has been moved forward to `xsep`, its semantics still need to be interpreted. An insight on the problem has been gained by carrying out experiments with the prototype system. In the prototype system, text processing was added as a post-processing stage. Creating a distinction between text and other musical features has previously been mentioned in this report as a needless complication.

The piece of music shown in Figure 29 was scanned and processed by the OMR system. The text was recognised and placed in the reconstructed page, however the positioning was not ideal. This is because the OMR system does not know what size the staves in Lime are drawn at, consequently it does not know what offset for text is proportionally pleasing. It is possible to generate Lime formatted files which specify the size of the staff, however it was felt that this conflicted with the goal of a computer pliable form of music. At this stage in the investigation of including text in a page of music, it is left to the user to use Lime to refine the position of the text. A similar point can be made for Figure 28, where the automatic layout of the reconstructed piece is cramped.

7 Miscellaneous

7.1 A Hymn Database

A database of on-line music was required for a different musical project. The idea behind the project was to match a whistled tune to a database of melodies. Hymn books often include a tune index of the melodies contained in the book. A suitable index was chosen and the existing OMR system was used to compile the database.

One of the pages from the hymn book is shown in Figure 31. The result of processing the image using the OMR system is shown in Figure 32. The most common mistakes are misclassified time-signatures and hollow note heads detected as full note heads. Because the index is for reference, the size of the music was smaller than ‘standard’ examples. That is not to say the music was poorly set or printed,

BREAK

The musical score for 'BREAK' is written for guitar in 4/4 time with a key signature of one sharp (F#). It consists of six systems, each with a treble staff and a three-string bass staff (T, A, B). The score includes various guitar techniques such as triplets, bends, and slides, indicated by numbers and symbols like 'H' (bend), 'P' (slide), and '7' (hammer-on). Chord symbols G(7), C, D7, and C7 are placed above the treble staff. The piece concludes with a double bar line on the sixth system.

35

Figure 27: Scanned original.

Lime print of "blues1_rc.lime" - "Score" at 16:54 on 11/24/94.

Figure 28: Reconstructed score using Lime.

1

CONCERTANTE

Allegro
(Solo)

London 1792
Hoboken I:105

Figure 29: Scanned piece, reduced to fit on page.

just that the level of detail captured in a 300 dpi scan was lower. For example, text in the scanned image frequently touched, despite the original characters being visibly separate. Rescanning the work at 600 dpi would rectify some of the mistakes.

In all, 36 pages were scanned to generate the database. So far, the mistakes for only the first 5 pages have been tabulated. The results appear in Table 3.

7.2 The IMS Handbook of Musical Codes

A book to be published is, “The IMS Handbook of Musical Codes”. The editors of the book acknowledge that a variety of musical file formats exist, and plan to publish a handbook of common codes to aid the translation between the various formats. The author is contributing a chapter on **Csound**. In the written chapter, BNF grammars are provided for the two file formats used by **Csound**. They have been implemented using *yacc* and are *ftp* available.

8 Reflections and Future Thoughts

Work has concentrated on developing algorithms. Little time, as yet, has been spent running experiments. This is an important element that will be rectified in the future. The resulting database of *prim* files will then prove a useful resource for the *collate* program.

ERROR: rangecheck
OFFENDING COMMAND: put

STACK:

CONCERTANTE

3 5 3 3
SAVIOUR CHRIST, 216

4 4 4 4 4 8
GARTAN, 5

4 8 8 4
ENIGMA, 106

4 9 4 8 9 9 4
THE INFANT KING (SING LULLABY), 92

5 5 5 5 5 5 4
HARROW WEALD, 2

5 5 5 5 D
VICTOR'S CROWN, 185

5 5 5 11
ARDWICK, 224

5 5 7 D
PASSFIELD, 382

5 5 8 D
SCHÖNSTER HERR JESU, 209

SILCHESTER, 209, 381

5 5 8 8 5 5
ARNSTADT (THURINGIA), 279, 302

5 6 6 4
SOMMERLIED, 393

TENHEAD, 546

vlu

Figure 31: Example page from hymn reference index.

3 5 3 3

SAV OUR CHRIST 216

4 GARTA 5 4 4 4 4 4 4 8

7 u IGUA 106 4 8 8 4

13 THE IN AN KI G SHI G I ULLABY 92 4 9 4 8 9 9 4

21 HARROWWEAL 2 5 5 5 5 5 5 5 4

25 VICTORS CROWN 185 5 5 5 5 1

29 ARDWICK 224 5 5 5 1 1

36 TASSFIEI 382 5 5 7 D

43 SCHONSTER HIERIGESU 2 5 5 8 1

48 SILCHESTER 209 381

53 ARNSTADT THURINGIA 279 3 2 5 5 8 8 5 5

59 SOMMERLIEI 393 5 6 6 4

63 TENHEAT 546

Lime print of "hymn47_c1.lime" - "Score" at 19:40 on 11/24/94.

Figure 32: Reconstructed page using Lime.

Classification Error	Page 1	Page 2	Page 3	Page 4	Page 5
Incorrect time-signature	8	10	11	6	6
Time-signature as note	1			5	11
Key-signature mismatch	1	1			4
Accidental mismatch/missing	2			2	3
Fragmented hollow note head				14	8
Hollow note head as full note head	5	4	9	12	20
Missing dot	2		1	2	4
Slur fragment as dot	1		2		
Missing slur				3	5
Slur as semibreve				2	2
Double bar as crotchet					
Incorrect number of beams					
Beam as chord					
Wrong pitch	2		1	1	
Incorrectly matched slur	1		1		1
Broken vertical line		1			
Semiquaver really quaver		1			
Missing semibreve		1		2	1
Missing quaver tail			1		
Number of of Objects	23	18	26	49	65
Recognition rate	91.67%	93.77%	92.38%	85.92%	83.95%

Table 3: Tabulated data for first 5 pages of hymn reference index.

Music often contains text which, for historical reasons, can be in one of a variety of languages. Music notation is based on different graphical layouts inferring different musical semantics. However, when considering text in different languages, there is the unusual property that different graphical layouts means the same thing. A good musical knowledge expression language would naturally encapsulate such trends. The point is made here as a reminder, so it is considered during the language design.

Robustness and reliability are critical attributes of any recognition system. They are a measure of success. It is believed the design philosophy for this OMR system reflects the importance of these criteria, whilst taking into consideration the ‘versatile foundation’ requirement, particular to OMR work.

References

- [Bai94a] David Bainbridge. *A Complete Optical Music Recognition System: Looking to the Future*. Technical report, Department of Computer Science, University of Canterbury, NZ, March 1994.
- [Bai94b] David Bainbridge. *Optical Music Recognition: Progress Report 1*. PhD thesis, Department of Computer Science, University of Canterbury, NZ, November 1994.
- [BFK⁺92] Thomas Bayer, Jurgen Franke, Ulrich Kressel, Eberhard Mandler, Matthias Oberlander, and Jurgen Schurmann. Towards the understanding of printed documents. In H. S. Baird, H. Bunke, and K. Yamamoto, editors, *Structured Document Image Analysis*. Springer, 1992.
- [BMW94] Timothy Bell, Alistair Moffat, and Ian Witten. *Managing Gigabytes: compressing and indexing documents and images*. Van Nostrand Reinhold, 1994.
- [BT88] R.D. Boyle and R.C. Thomas. *Computer Vision: A First Course*. Artificial Intelligence Texts. Blackwell Scientific Publications, 1988.
- [FB92] H. Fahmy and D. Blostein. Graph grammar processing of uncertain data. In H. Bunke, editor, *Advances in Structural and Syntactic Pattern Recognition (Proceedings of International Workshop on Structural and Syntactic Pattern Recognition, Bern, CH)*, volume 5 of *Series in Machine Perception and Artificial Intelligence*, pages 373–382. World Scientific, 1992.
- [FvDFH90] James D. Foley, Andries van Dam, Steven K. Feiner, and John F. Hughes. *Computer Graphics: Principles and Practice*. Addison-Wesley Publishing Company, second edition, 1990.
- [GP87] Rafael C. Gonzalez and Wintz P. *Digital Image Processing*. Addison-Wesley Publishing Company, 1987.
- [HB92] Lippold Haken and Dorothea Blostein. The tilia music representation: Extensibility, abstraction, and notation contexts for the line editor. In *Draft copy for Intensive Workshop in Sound Computation*, 1992.
- [MB91] P. Martin and C. Bellissant. Low-level analysis of music drawing images. In *Proceedings of First International Conference on Document Analysis*, volume 1, pages 417–425, Saint-Malo, France, 1991.
- [Rum90] Francis Rumsey. *MIDI Systems and Control*. Focal Press, 1990.